TEMA 10: OPTIMIZACIÓN DE CÓDIGO.

Bloque básico (BB): conjunto de instrucciones que se ejecutan sin posibilidad de bifurcaciones.

10.1 ALGORITMO PARA DIVIDIR TODO EL CONJUNTO DE INSTRUCCIONES EN BLOQUES BÁSICOS.

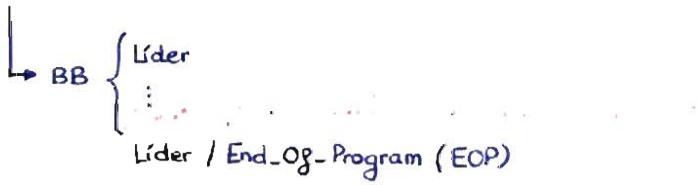
Algoritmo:

Entrada: Secuencia de proposiciones de tres direcciones (Código Intermedio)

Salida: Lista de bloques básicos (BBs)

Procedimiento:

1. Determinar conjuntos de líderes (primera proposición de un bloque básico) mediante las siguientes reglas:
   a) La primera proposición es líder.
   b) La proposición destino de un salto (condicional o incondicional) es líder.
   c) La proposición inmediatamente después de un salto (condicional o incondicional) es líder.
2. Para cada líder, su bloque básico consta del líder y las proposiciones hasta el siguiente líder (excluido éste) o hasta el fin del programa.



Ejemplo: Cálculo de bloques básicos.

```
(1)   a := 5          ← Líder (a)        } BB1
(2)   t1 := a + 2                        }
(3)   b := t1                            }
(4)   if b > 30 goto (10)  ← Líder (b)   } BB2
(5)   t2 := b - a     ← Líder (c)        } BB3
(6)   b := t2                            }
(7)   t3 := a - 1                        }
(8)   a := t3                            }
(9)   goto (4)                           }
(10)  t4 := b * a     ← Líder (b)(c)     } BB4
(11)  b := t4                            }
```

10.2. TRANSFORMACIONES EN EL CÓDIGO.

Pueden aplicarse una serie de transformaciones a un Bloque Básico para obtener otro Bloque Básico equivalente con mejor calidad.

Las transformaciones posibles son:

① Eliminar subexpresiones comunes:

```
a := b + c
b := a - d
c := b + c
d := a - d   => d := b
```

② Eliminar código inactivo (un nombre no se vuelve a usar):

```
a := b + c
e := a - d
c := a * b
   /* No aparece e */
```

Se puede eliminar "e := a - d" porque la variable "e" no se vuelve a utilizar.

③ Renombramiento de variables temporales:

```
t := a + b          u := a + b
  ...        =>       ...
d := t * 2          d := u * 2
```

"u" se usa antes o después. Se reutiliza la posición de memoria de "u" en lugar de crear un nuevo temporal "t".

④ Intercambio de proposiciones:

```
a := b + c          a := b + c
d := e * t   =>     f := a + b
f := a + b          d := e * t
```

"a" está en un registro.

⑤ Transformaciones algebraicas:

```
t := t + 0
t := t * 1   } Instrucciones inútiles
t := x ** 2  =>  t := x * x
ADD #1, R    =>  INC R
```

Instrucciones equivalentes más eficientes.